Third year

Communications

Data structures

(Second term 2011-2012)

(16)

(Final Revision – Part II)

Price: "7" L.E

Tel. no.: 010-03654726

Cairo University
Dept. of Electronics & Communications Engineering
Third Year
Maximum grade is 70 points

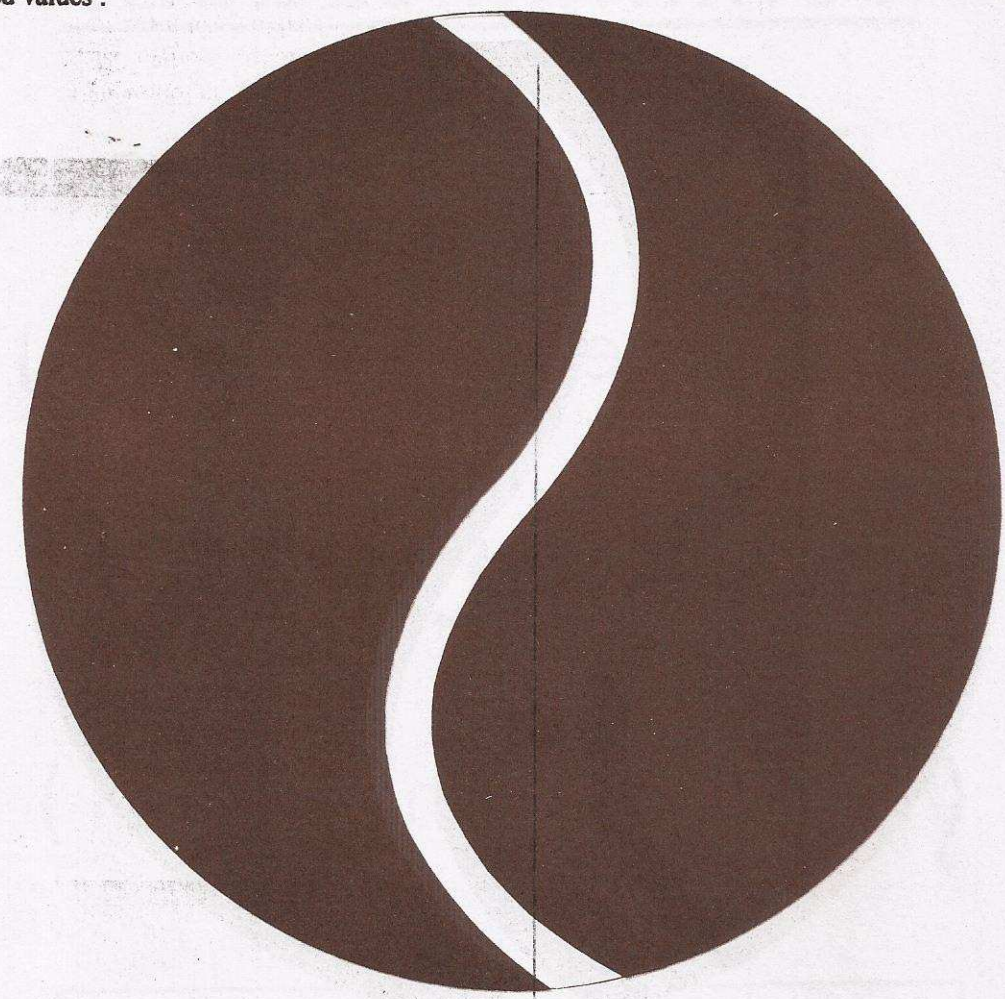
Faculty of Eng...
M...
Comp...
Time: 2 ...

أجب على الأسئلة الآتية
يرجى التأكد من وجود ٦ أوراق بالكراسة مشتملة صفحة بيضاء فى نهاية الكراسة
فى حالة عدم كفاية المساحة المخصصة لإجابة أحد الأسئلة، أكمل الأجابة فى الصفحة المقابلة له أو فى المساحة المخصصة نهاية الكراسة

Question #1 (Total 12 points)

Define and implement the class `sq_matrix` which represents a square matrix of doubles. The dimension of ... matrix is less than 100. The class supports the following operations: (4 points for proper class declaration)

- (2 points) `square_matrix(unsigned int dimn)` : it sets the dimension of the matrix to `dimn` and ... the elements are initialized to zeros. It must check that `dimn` is less than 100.
- (3 points) `Add(const matrix &B)` : which adds matrix `B` to `current` matrix. The operation is only v... if the two `matrices are of the same dimension`.
- (3 points) `Copy(const matrix &B)` : which copies matrix `B` to `current` matrix overriding dimension ... stored values .



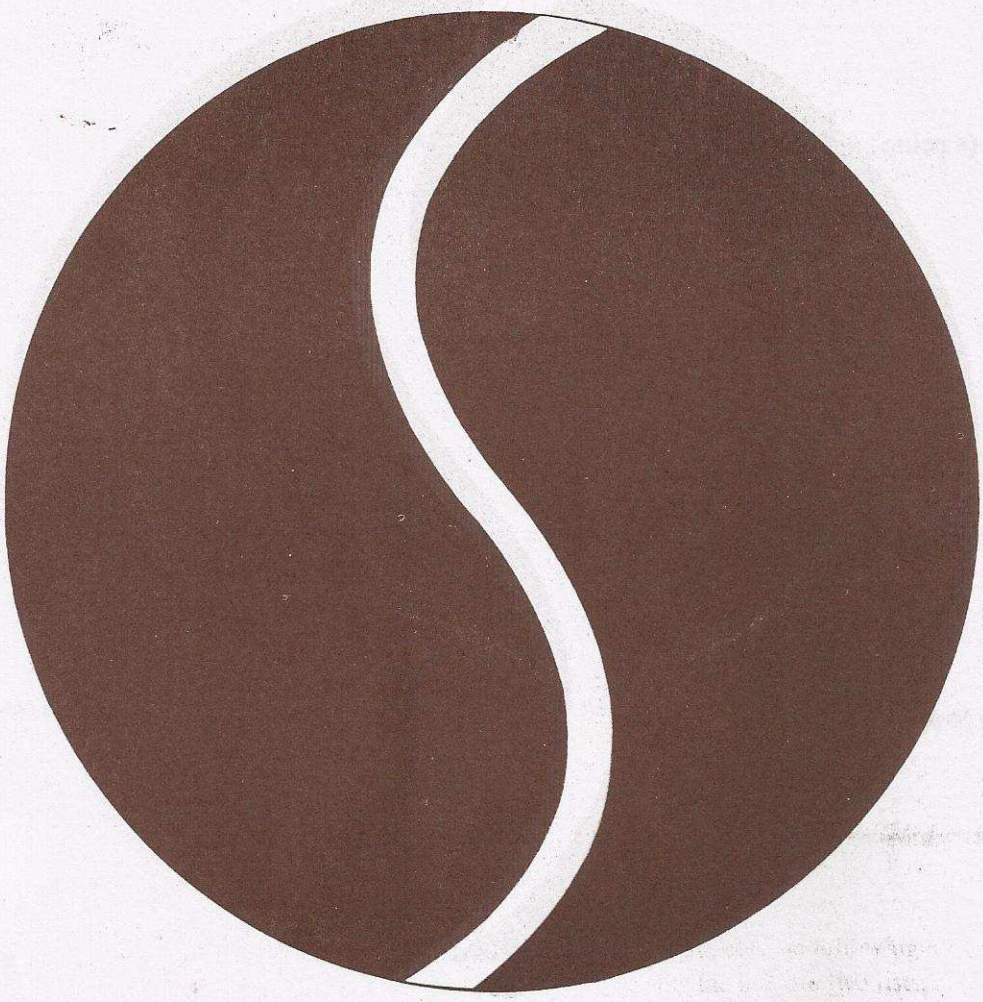
1

Question #2 (Total 20 points)

Using the standard singly linked-list storing items of `ListElemType`, add the following member functions:

- `Split(ListElemType pivot, List &L2)` which splits the list into two lists, all elements smaller than pivot remain in the current list and all elements greater than or equal to pivot move to list L2.
- `Sort( )`: This function sorts the current list using the **selection sort** algorithm.

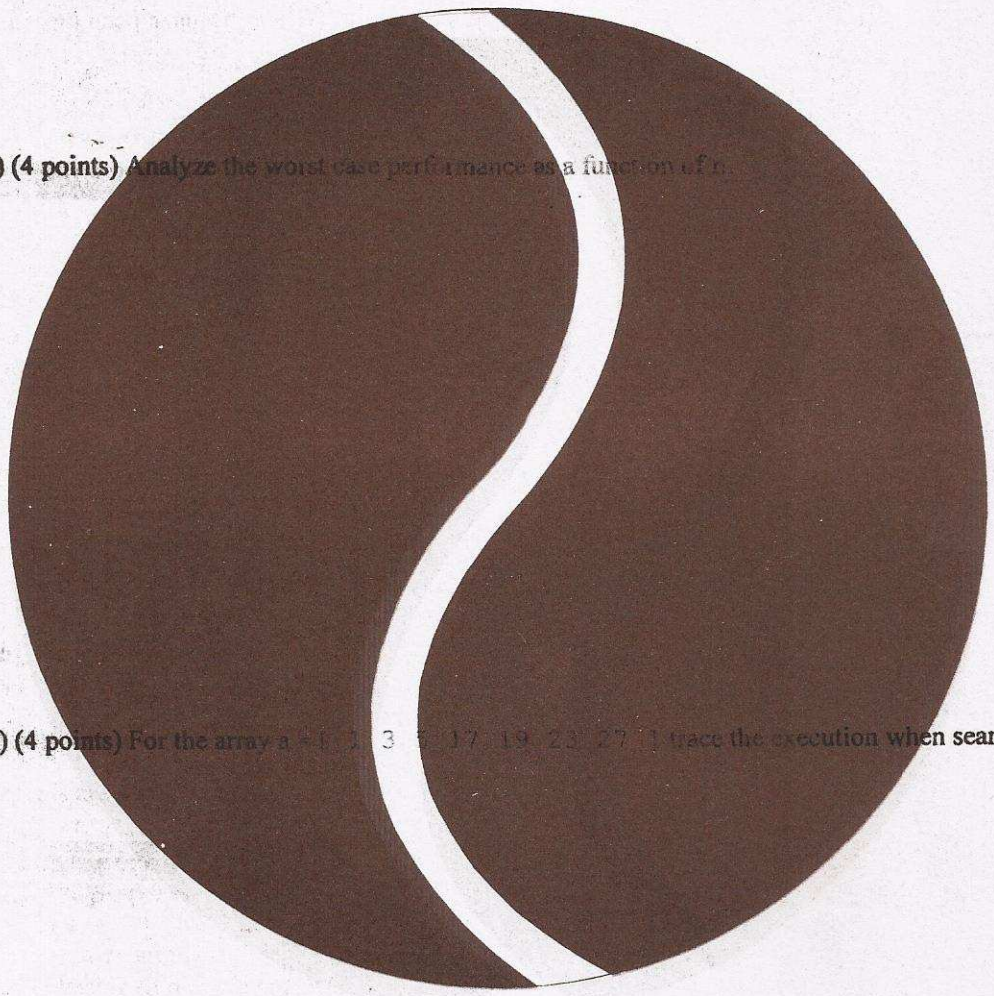
Write the definition of the class (4 points) and write the code of the above two member functions (8 points each)



2

Question#3 (Total 14 points)

a) (6 points) Write the code for binary search algorithm that searches a sorted array a [ ] of integers for the ele
equal to key. The array size is passed as parameter n.



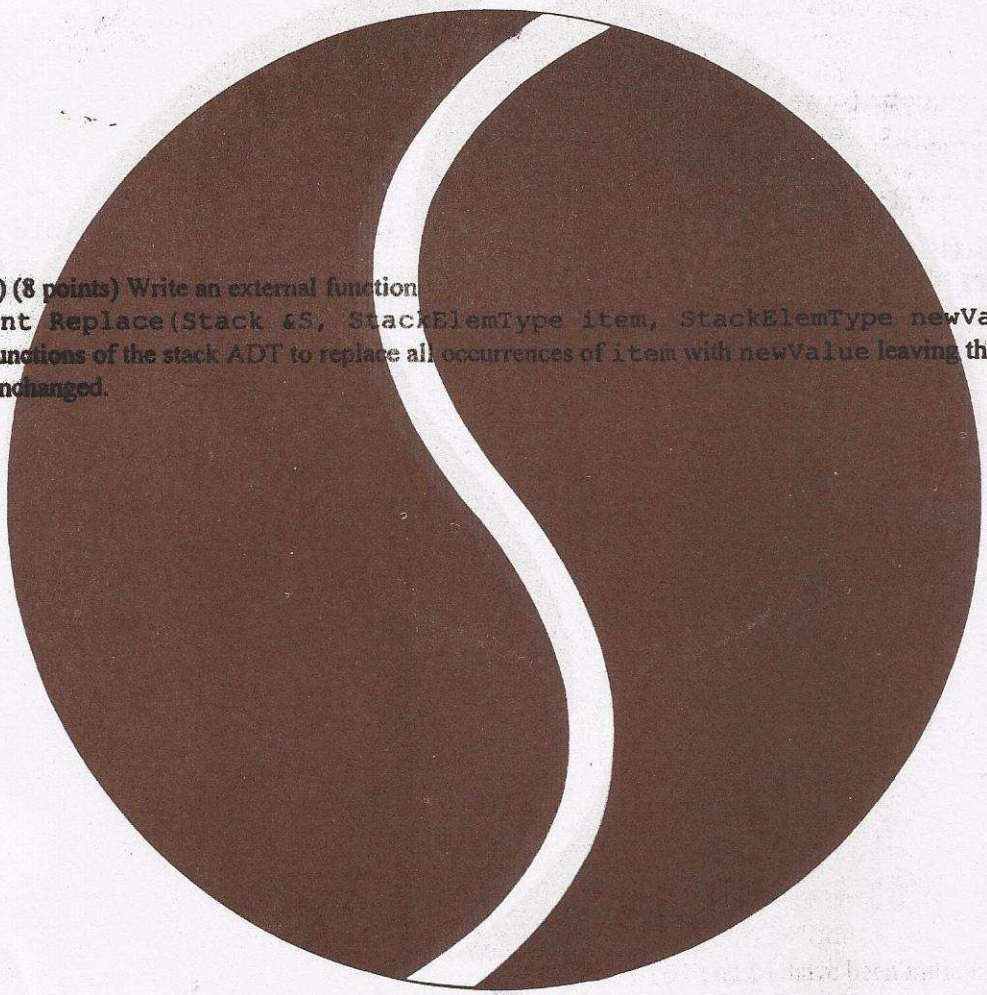
b) (4 points) Analyze the worst case performance as a function of n.

c) (4 points) For the array a = { 1, 3, 5, 17, 19, 23, 27 } trace the execution when searching for 24.

3

Question#4 (Total 16 points)

a) (8 points) Write an external function `Boolean Identical(const Queue &Q1, const Queue &Q2)` that determines if the two queues are identical. Preconditions: `Q1` and `Q2` have been initialized. Postconditions: Queues are unchanged and function return value is true if the two queues are identical, false otherwise.



b) (8 points) Write an external function `int Replace(Stack &S, StackElemType item, StackElemType newValue)` that uses member functions of the stack ADT to replace all occurrences of `item` with `newValue` leaving the rest of the stack unchanged.

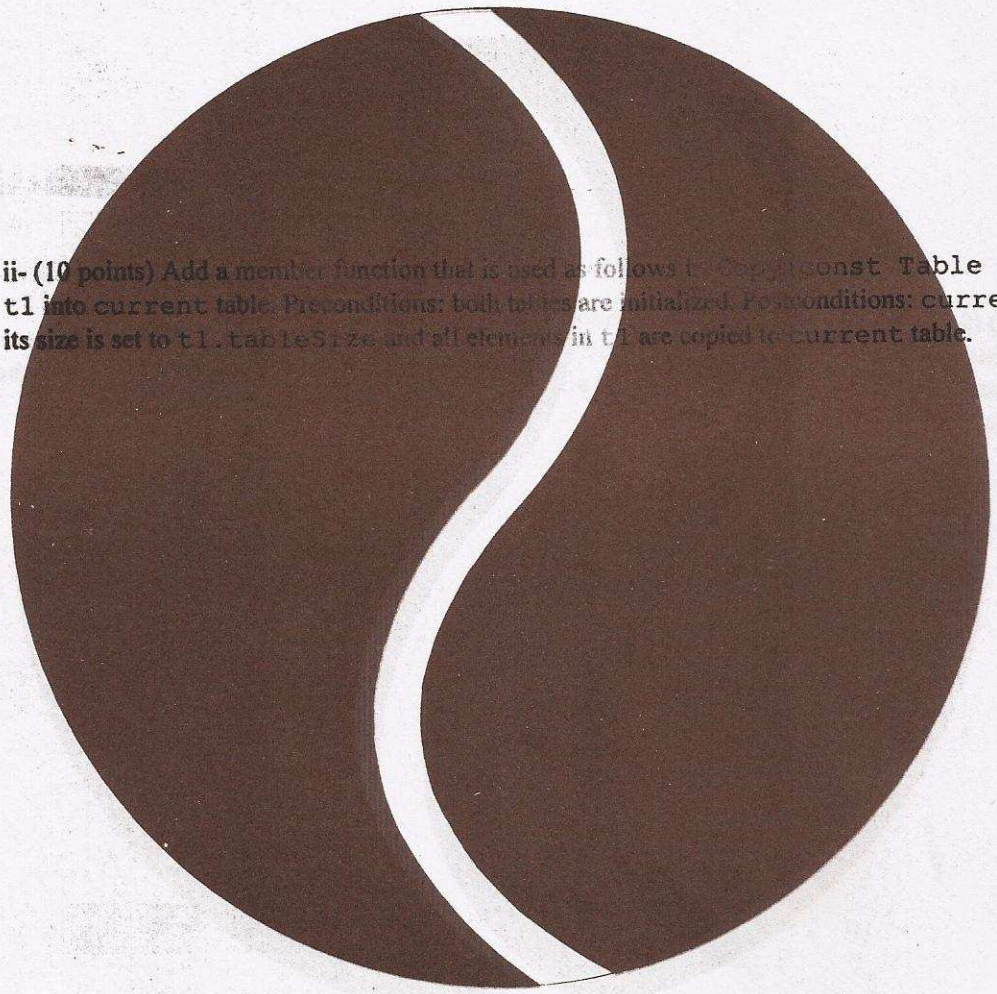
i

4

Question#5 (Total 18 points)

Consider the implementation of the chained hash tables, with the standard operations of lookup, deleteKey, and i
The entries in the table is kept in an array with maximum size `MAX_TABLE_SIZE`, however, the actual table siz
can be smaller than `MAX_TABLE_SIZE` and is kept in a private variable called `tableSize` that is passed as
parameter to the constructor, i.e. the constructor of the Table class will look as follows:
`Table::Table(int t_size.` For simplicity you may disregard the usage of templates for this question.

i- (6 points) Write the code for the new constructor function which should assure that
`t_size <= MAX_TABLE_SIZE`. If `t_size` is equal to zero, the table size becomes `MAX_TABEL_SIZE`.



ii- (10 points) Add a member function that is used as follows: `Copy(const Table &t1)` that copies the t
`t1` into `current` table. Preconditions: both tables are initialized. Postconditions: `current` table is erased and
its size is set to `t1.tableSize` and all elements in `t1` are copied to `current` table.

Cairo University
Dept. of Electronics & Communications Engineering
Third Year
Maximum grade is 70 points

Faculty of Engineering
May 2010
Computers 3B
Time: 2 hours

اجب على الأسئلة الآتية

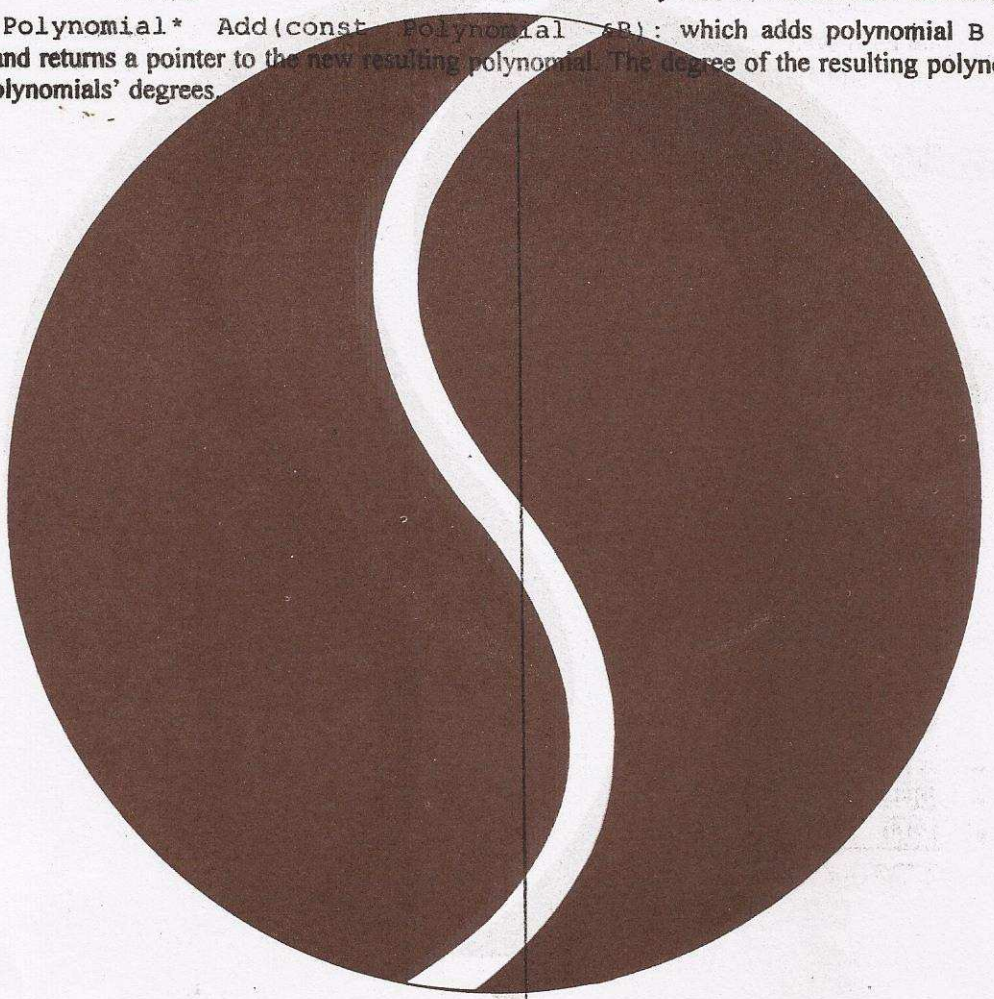
يرجى التأكد من وجود ٦ أوراق بالكراسة مشتملة صفحة بيضاء في نهاية الكراسة

في حالة عدم كفاية المساحة المخصصة لإجابة أحد الأسئلة، أكمل الأجابة في الصفحة المقابلة له أو في المساحة المخصصة نهاية الكراسة

Question #1 (Total 15 points)

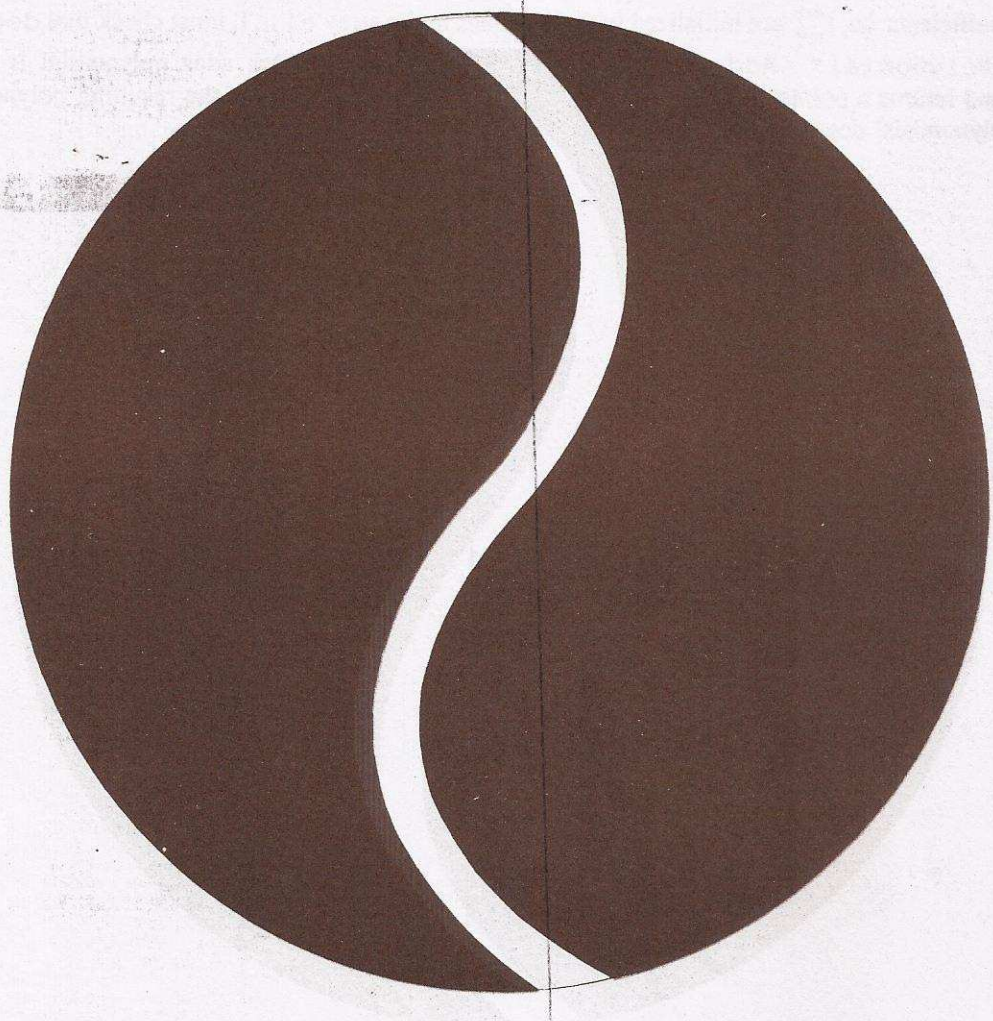
Define and implement the class `Polynomial` which represents polynomial of the form $P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0$. The degree of the polynomial n is less than 100. The class supports the following operations: (4 points for proper class declaration)

- (2 points) `Polynomial (unsigned int deg)`: it sets the degree of the polynomial to deg and all the coefficients $\{a_i\}_{i=0}^{deg}$ are initialized to zeros. It must check that deg is less than 100.
- (3 points) `Polynomial (unsigned int deg, double a[])`: it sets the degree of the polynomial to deg and all the coefficients $\{a_i\}_{i=0}^{deg}$ are initialized with the values in the array `a[]`. It must check that deg is less than 100.
- (5 points) `Polynomial* Add(const Polynomial &B)`: which adds polynomial B to the current polynomial and returns a pointer to the new resulting polynomial. The degree of the resulting polynomial is the larger of the two polynomials' degrees.



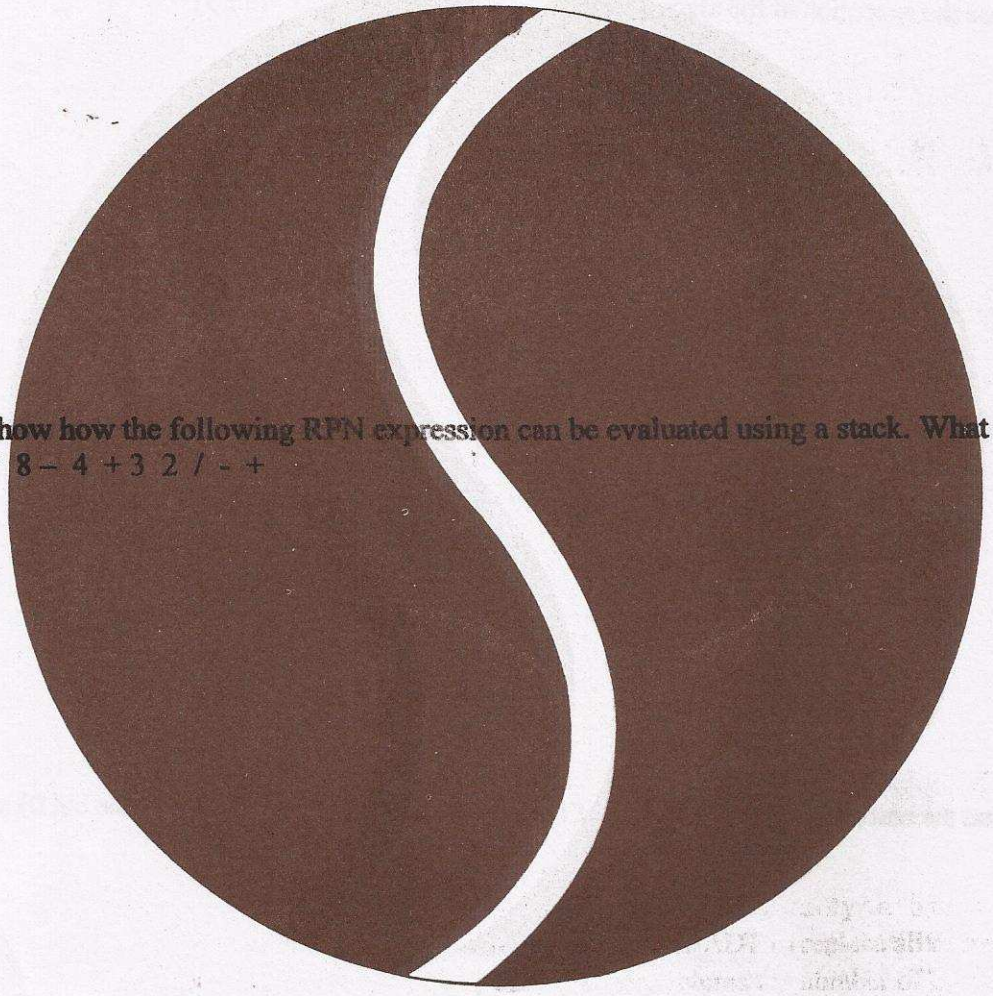
Question #2 (Total 18 points)

Define a single-linked circular list (CircList) ADT supporting the standards list operations of `First`, `Next`, `Insert`. In addition the list supports the operations `void Merge(const CircList &clist)` which merges `clist` with current list and the operation `ReversePrint` which prints the elements in the list in reverse order without using any additional structures. Write the definition of the class and the functions `Insert`, `Merge`, and `ReversePrint`.



Question 3 (Total 12 points)

b) (6 points) Write an external function `int Replace(Queue &Q, QueueElemType item, QueueElemType newValue)` **that uses member functions of the queue ADT to replace all occurrences of** `item` **with** `newValue` **leaving the rest of the queue unchanged. The functions returns the number of queue entries that were replaced.**



b) (6 points) Show how the following RPN expression can be evaluated using a stack. What is the final result? 5 3 * 8 – 4 + 3 2 / - +

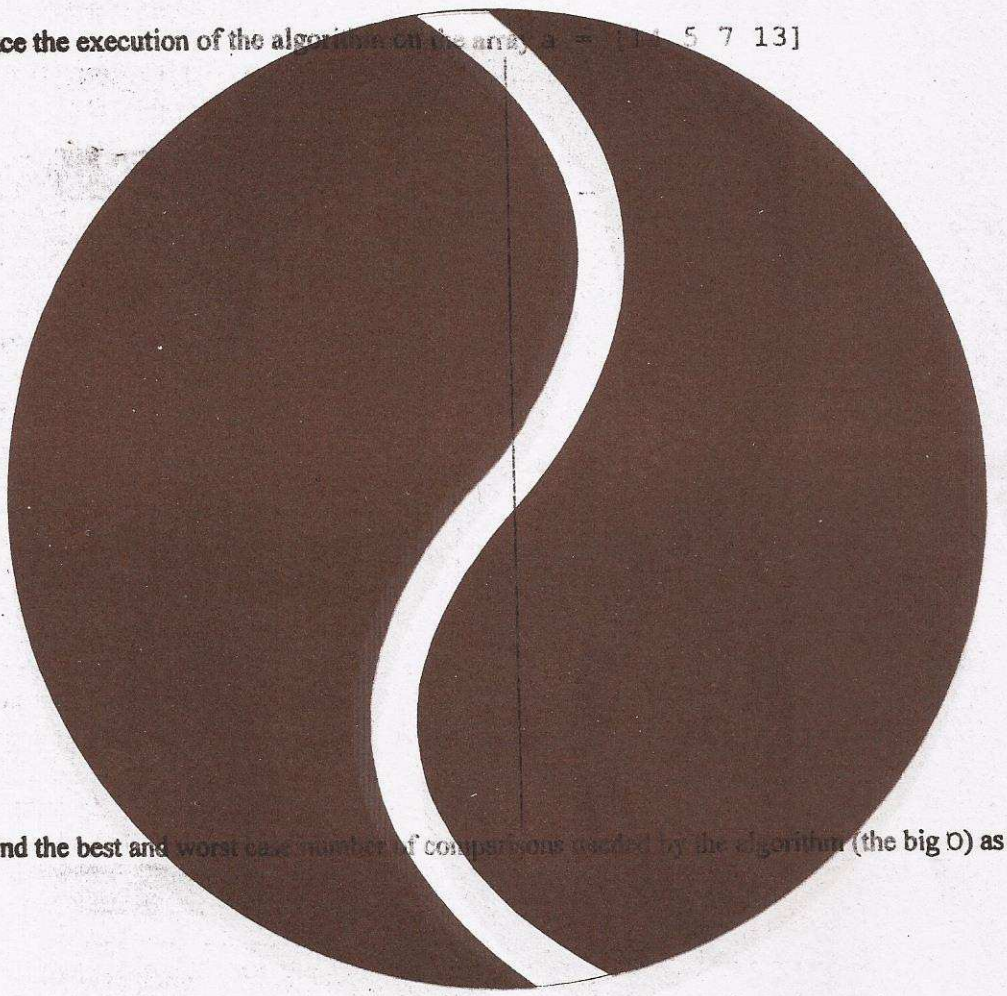
[8]

Question#4 (Total 14 points)

The following is an algorithm known as `InsertSort` that sorts array a in ascending order.

```
void insertionSort(int a[], int n)
{
      int x; // item in the array
      int i, j;
      for (i=1; i < n; i++){
            x = a[i];
            j = i - 1;
            while (j >=0 && a[j] > x){
                  a[j+1] = a[j];
                  j--;
            } // end while
            a[j+1] = x;
      } // end for
}
```

a) [8 points] Trace the execution of the algorithm on the array a = [ ] 5 7 13]



b) [6 points] Find the best and worst case number of comparisons needed by the algorithm (the big O) as function of n (array size).

Best case:

Worst case:

a) [10 points] Classify the following statements as T or F, with brief explanation why.

i) **Binary search is an O(n log n) algorithm.**

ii) **Binary search is always faster than sequential search.**

iii) **When hashing is used increase size of hash table array always reduces the number of collisions.**

iv) **If we use chaining in a hashing scheme we don't have to worry about collision resolution.**

v) **For population size of 50, a table size of 101 (a prime number) will always provide better collision performance if table size is 100.**

b) [10 points] Consider the keys 66 47 87 90 126 140 145 153 177 285 393 395 467 566 620 735. Show how these values will be stored in a hash table:

i) **Using linear probing with a hash table of size 20.**

ii) **Using chained hashing with a table of size 10.**

Then fill the provided table indicating the number of comparisons needed to search for the given key.

Linear Probing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	

Chained Hashing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

#of comparisons

Key	Linear Probing	Chained Hashing
87		
566		
153		

10

...nics & Communications Engineering

...e is 70 points

أجب على الأسئلة الآتية

يرجى التأكد من وجود ٦ أوراق بالكراسة مشتملة صفحة بيضاء فى نهاية الكراسة

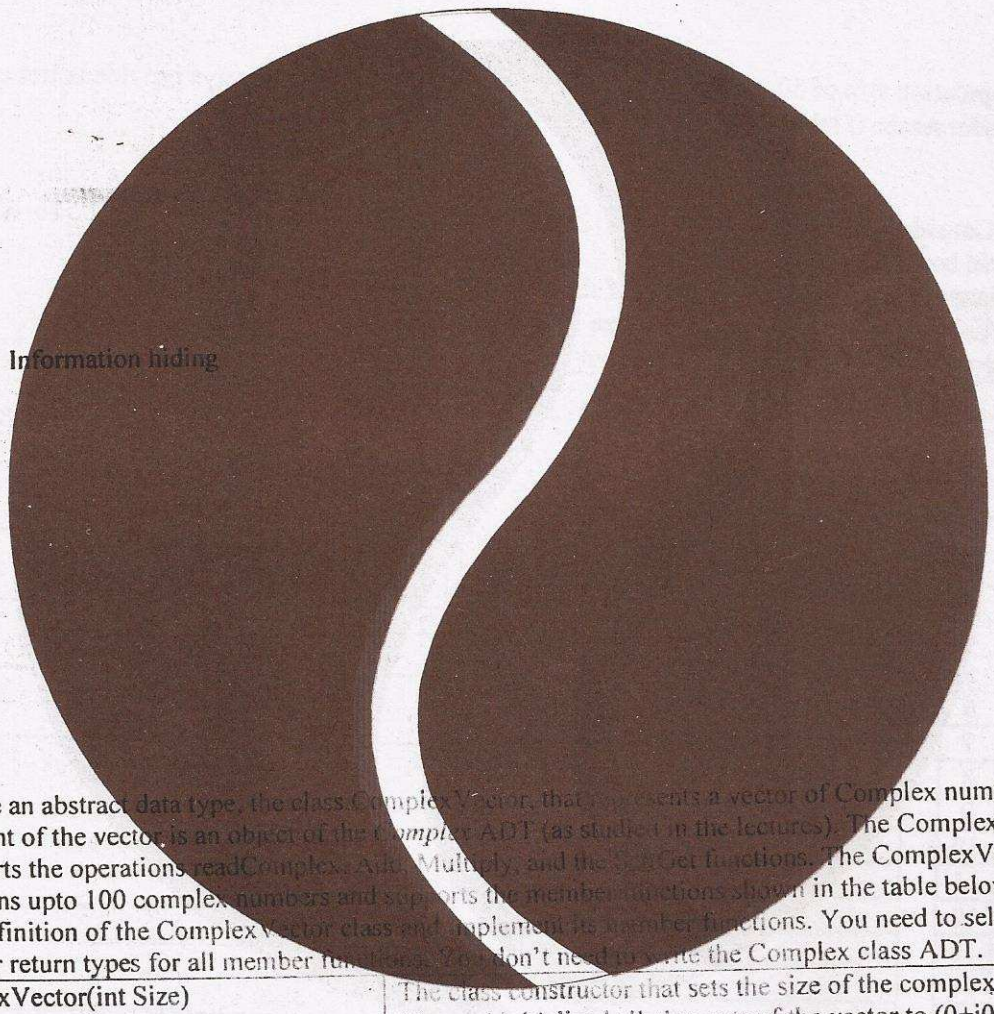
فى حالة عدم كفاية المساحة المخصصة لإجابة أحد الأسئلة، أكمل الأجابة فى الصفحة المقابلة له أو فى ال...

المخصصة نهاية الكراسة

...#1 (20 points)

...e briefly what is meant by the following terms and their role in Object-Oriented Design and Programming:

Abstraction



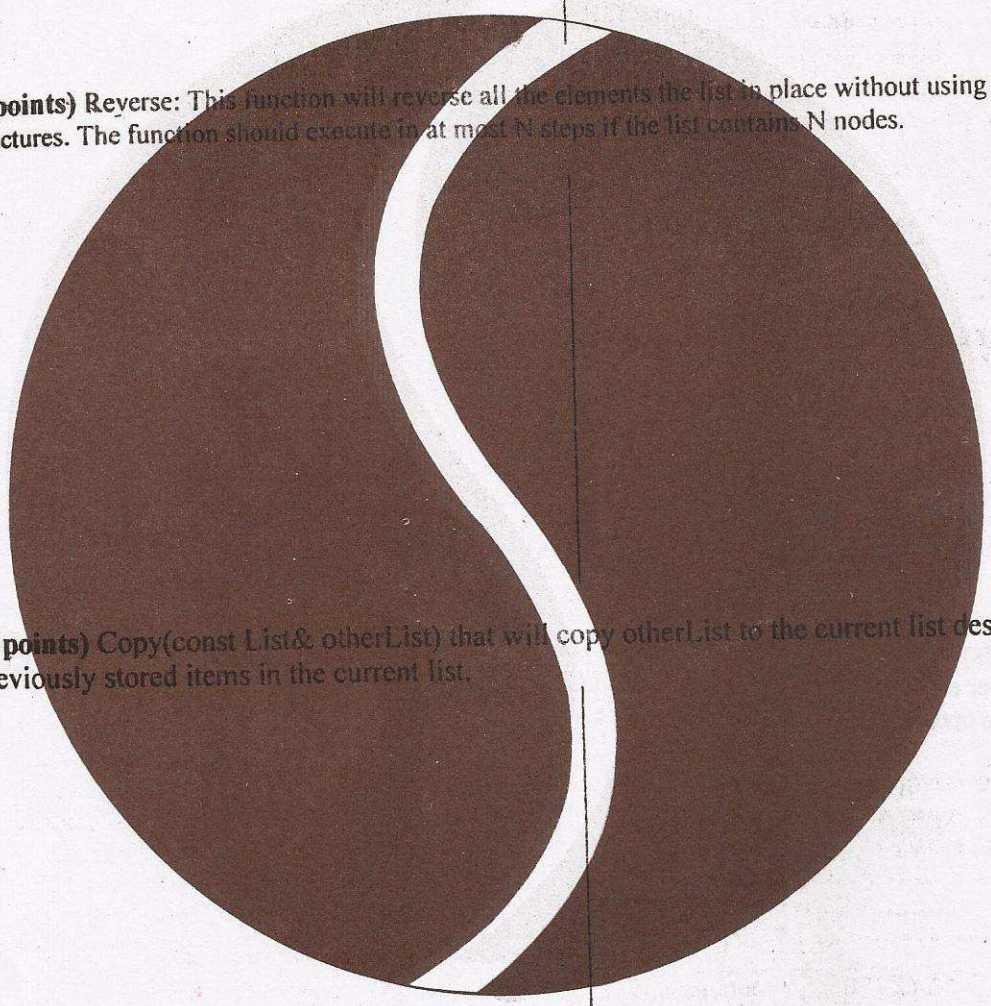
...Information hiding

...Define an abstract data type, the class ComplexVector, that represents a vector of Complex numbers. Each element of the vector is an object of the Complex ADT (as studied in the lectures). The Complex ADT supports the operations readComplex, Add, Multiply, and the Set/Get functions. The ComplexVector class contains upto 100 complex numbers and supports the member functions shown in the table below. Write the definition of the ComplexVector class and implement its member functions. You need to select the proper return types for all member functions. You don't need to write the Complex class ADT.

...mplexVector(int Size)	The class constructor that sets the size of the complex vector to Size and initialized all elements of the vector to (0+j0).
...etElem(int i, const Complex	Sets the value of element *i* of the vector to *c*. It should check that *i* is less than size of the vector.
...Add(const ComplexVector &x)	Adds the vector *x* to the *current* object. The function should first assert that the two vectors have the same size, if so the addition is performed.
...DotProduct(const ...plexVector &x)	Calculates the dot product of the vector *x* and the *current* object. The function should first assert that the two vectors have the same size, if so the addition is performed.

a standard single-linked list ADT storing items of ListElemType. Write code for the following member

oints) Interchange: This function will consider each two adjacent nodes together and exchange the data in So, starting with the head as node 0, the contents of nodes 0 and 1 will be exchanged, nodes 2 and 3 will be nged and so on. If the list contains an odd number of nodes, the contents of the last node will not be nged with any other node.



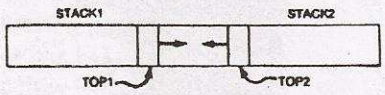
(6 points) Reverse: This function will reverse all the elements the list in place without using any additional structures. The function should execute in at most N steps if the list contains N nodes.

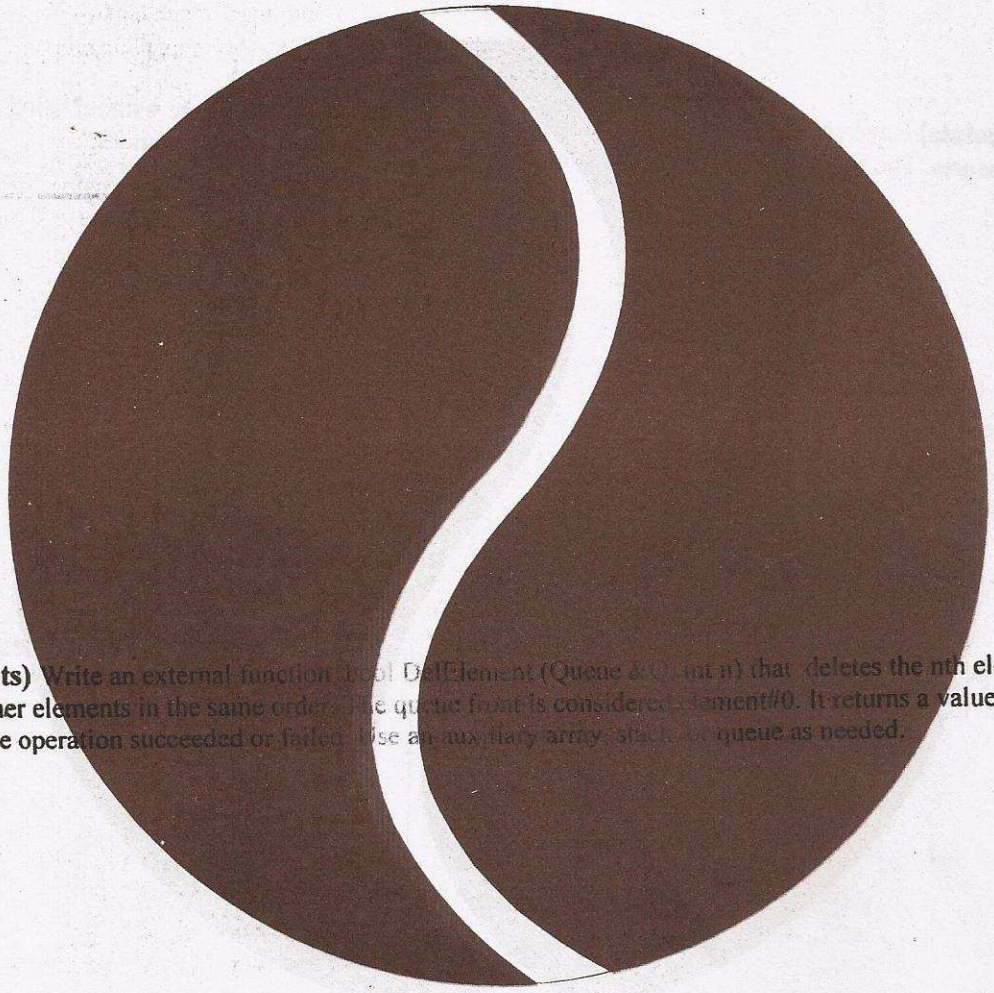
(8 points) Copy(const List& otherList) that will copy otherList to the current list destroying all previously stored items in the current list.

12

(16 points)

points) Two stacks are implemented in one array of length 100 as shown. The tops of the stacks are pointed to
top2 and grow toward each other. Write a procedure int
DataType v, int s) that pushes the contents of v onto the stack indicated by s.
a value of 1 or 2. If the stack to which v is to be added is full, then push
add it, but should return -1, if push succeeds it returns s. At initialization
nd top2 = 100. Provide the condition for a full stack.





6 points) Write an external function bool DelElement (Queue &Q, int n) that deletes the nth element of a queue,
ving other elements in the same order. The queue front is considered element#0. It returns a value indicating
ther the operation succeeded or failed. Use an auxiliary array, stack or queue as needed.

131

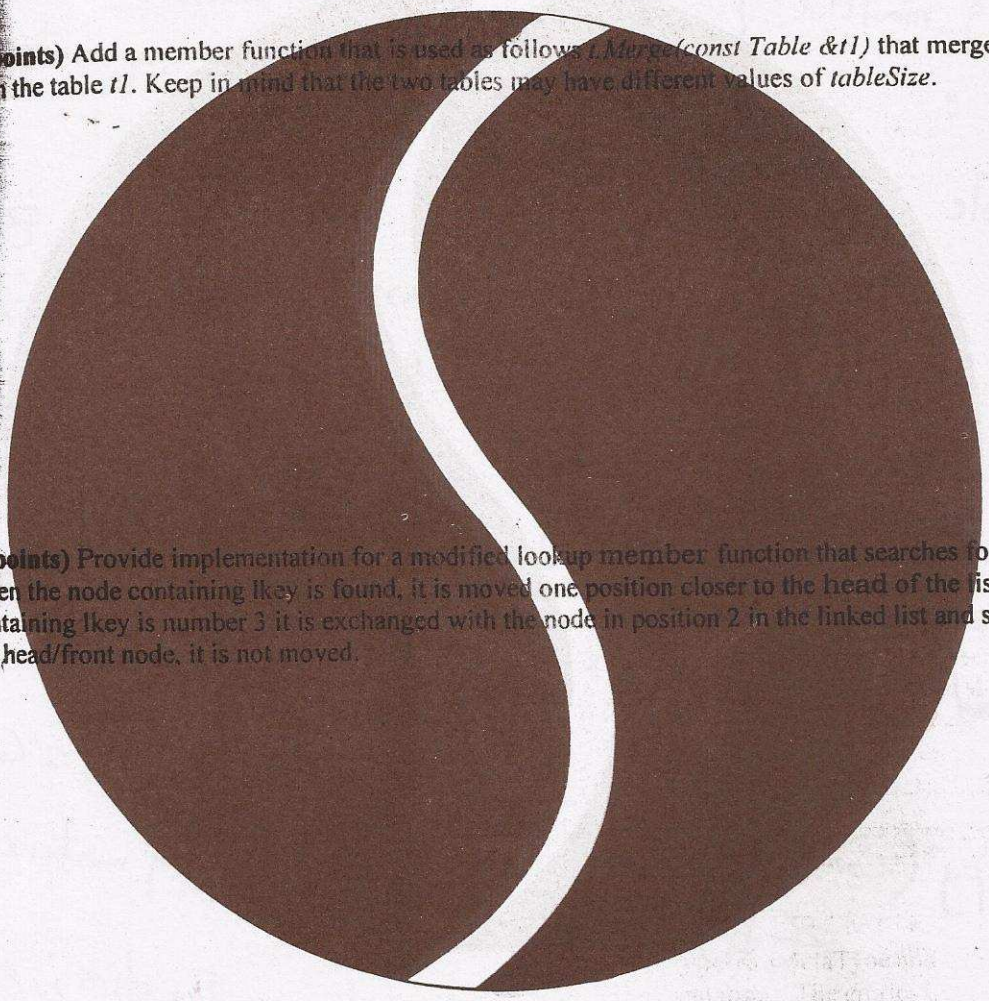
Canceled & solved before

(20 points)

e implementation of the chained hash tables with keys of type KeyDataType and associated data of pe, supporting the standard operations lookup, deleteKey, and insert. The entries in the table is kept in an maximum size *MAX_TABLE_SIZE*, however, the actual table size can be smaller than *MAX_TABLE_SIZE* in a private variable called *tableSize* that is passed as parameter to the constructor, i.e. the constructor of lass will look as follows: *Table::Table(int t_size)*. For simplicity you may ignore templates usage for this

nts) Write the code for the new constructor function. Make sure that *t_size* <= *MAX_TABLE_SIZE*.

points) Add a member function that is used as follows *t.Merge(const Table &t1)* that merges the current table *t* th the table *t1*. Keep in mind that the two tables may have different values of *tableSize*.



points) Provide implementation for a modified lookup member function that searches for key lkey. If and when the node containing lkey is found, it is moved one position closer to the head of the list (so if the node containing lkey is number 3 it is exchanged with the node in position 2 in the linked list and so on). If the node is the head/front node, it is not moved.

19

Final 2009:

Question 1:

```
Const int maxsize = 100;
class sq_matrix {
 private:
   double A[maxsize][maxsize];
   int dim;
 public:
   sq_matrix (unsigned
     int dimn);
   Void Add (Const sq_matrix
     & B);
   Void Copy (Const sq_matrix
     & B);
};

sq_matrix::sq_matrix
    (unsigned int dimn)
{ assert (dimn < maxsize);
  dim = dimn;
  for (int i=0; i<maxsize; i++)
    for (int j=0; j<maxsize; j++)
      A[i][j] = 0;
}

sq_matrix::Add
    (Const sq_matrix & B)
{ assert (dim == B.dim);
  for (int i=0; i<dim; i++)
    for (int j=0; j<dim; j++)
      A[i][j] += B.A[i][j];
}

sq_matrix::Copy
    (Const sq_matrix & B)
{ dim = B.dim;
  for (int i=0; i<dim; i++)
    for (int j=0; j<dim; j++)
      A[i][j] = B.A[i][j];
}
```

Final 2009:

Question 1:

```
Const int maxsize = 100;
class sq_matrix {
 private:
    double A[maxsize][maxsize];
    int dim;
 public:
    sq_matrix (unsigned int dimn);
    Void Add (Const sq_matrix &B);
    Void Copy (Const sq_matrix &B);
};

sq_matrix :: sq_matrix
        (unsigned int dimn)
{ assert (dimn < maxsize);
  dim = dimn;
  for (int i=0; i<maxsize; i++)
    for (int j=0; j<maxsize; j++)
      A[i][j] = 0;
}

sq_matrix :: Add
        (Const sq_matrix &B)
{ assert (dim == B.dim);
  for (int i=0; i<dim; i++)
    for (int j=0; j<dim; j++)
      A[i][j] += B.A[i][j];
}

Void sq_matrix :: Copy
        (Const sq_matrix &B)
{ dim = B.dim;
  for (int i=0; i<dim; i++)
    for (int j=0; j<dim; j++)
      A[i][j] = B.A[i][j];
}
```

Question 2:

* class definition:

```
typedef int listElemType;
class list {
  private:
    struct node;
    typedef node* link;
    struct node {
      listElemType elem;
      link next; };
    link head; link current;
  public:
    list ();
    bool first (listElemType &elem);
    " next ( " &elem);
    void insert ( " elem);
    void split (listElemType
      pivot, list & L2);
    void Sort ();
};
```

* void list::split (listElemType elem, list & L2)

```
{
  link del;
  while ( head &&
      head->elem >= pivot)
  { L2.insert (head->elem);
    del = head;
    head = head->next;
    delete del;
  }
  link pred = head;
  while (pred && pred->next)
  {
    if (pred->next->elem
        >= pivot)
    { L2.insert (pred->next->
        elem);
      del = pred->next;
      pred->next = del
        ->next;
      delete del;
    }
    else
      pred = pred->next;
  }
}
```

```
void list::sort ()
{
  link minpos, p; listElemType temp;
  for (link ptr=head; ptr != ptr->next;
       ptr = ptr->next)
  {
    minpos = ptr;
    for (p = minpos; p->next; p=p->next)
      if (minpos->elem > p->next->elem)
        minpos = p->next;

    temp = minpos->elem;
    minpos->elem = ptr->elem;
    ptr->elem = temp;
  }
}
```

op for
items in list

search for min

swap in with first elem

Question 3:

- a & b : see notes
- c) first step:

first = 0 , last = 6 ,

$$mid = \frac{0+6}{2} = 3$$

a[3] < key

17 24

first = mid + 1 = 4

Second step:

- first = 4, last = 6, mid = $\frac{4+6}{2} = 5$
- a[5] < key

23 24

first = mid + 1 = 6

Third step:

- first = 6, last = 6, mid = $\frac{6+6}{2} = 6$
- a[6] > key

27

last = mid - 1 = 5

Fourth step: first > last

"item is not found"

Question 4:

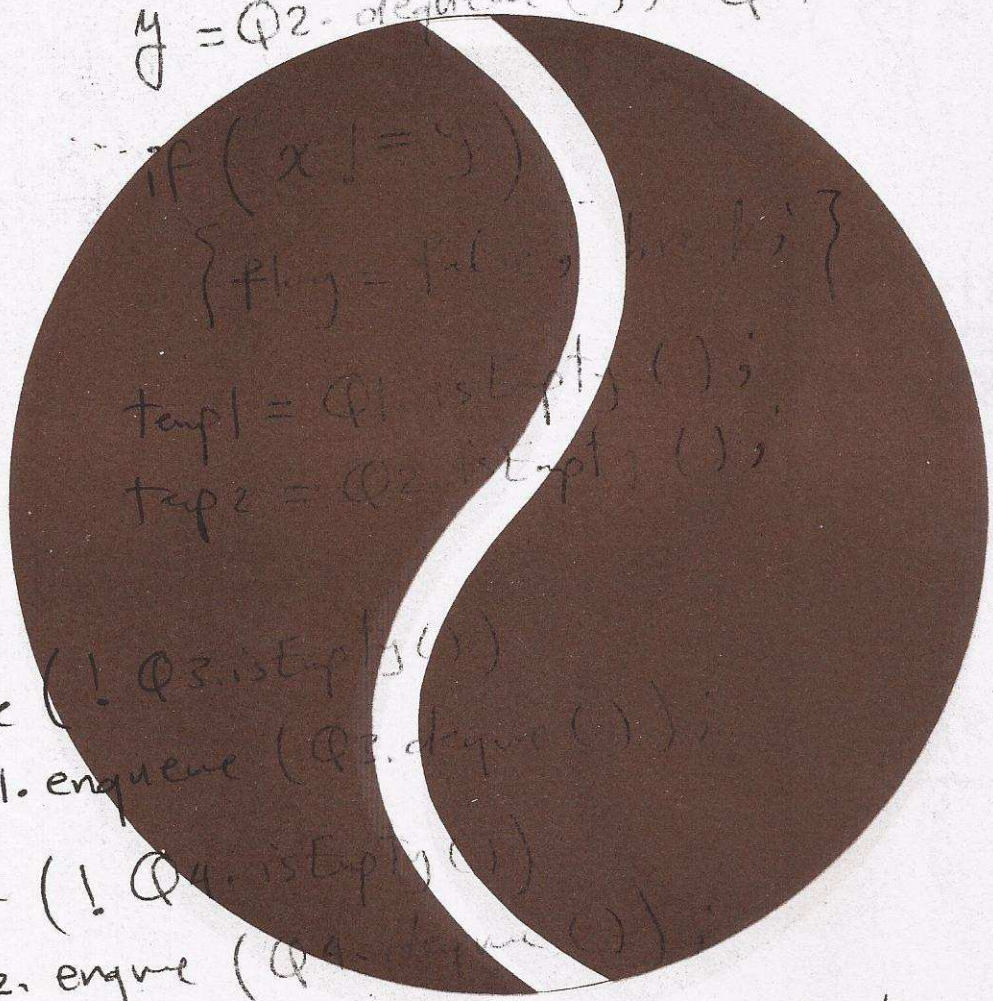
[a] bool identical (const Queue & Q1, const Queue & Q2)

{

Queue Q3, Q4;

QueueElemType x, y; bool flag = true;

```
bool temp1 = Q1.isEmpty();
bool temp2 = Q2.isEmpty();
while (!temp1 && !temp2)
{
    x = Q1.dequeue();  Q3.enqueue(x);
    y = Q2.dequeue();  Q4.enqueue(y);
    if (x != y)
    { flag = false; break; }
    temp1 = Q1.isEmpty();
    temp2 = Q2.isEmpty();
}
while (!Q3.isEmpty())
    Q1.enqueue(Q3.dequeue());
while (!Q4.isEmpty())
    Q2.enqueue(Q4.dequeue());
if ((!temp1 && !temp2) || temp1 || temp2)
    flag = false;
return flag;
}
```



(b) Solved before.

Question 5:

i- solved before.

ii-

```
void Table::Copy(const Table &t1)
{
    link del, P;
    for (int i=0; i<tablesize; i++)
        for (P=T[i]; P; )
            { del = P;
              P=P->next; delete P; }
    tablesize = t1.tablesize;
    for (int i=0; i<tablesize; i++)
        for (P=t1.T[i]; P; P=P->next)
            insert(P->key, P->data);
}
```

Final 2010

Question 1:

```
Const int N=100;
class Polynomial {
private:
    double Arr[N];
    int n;
public:
    Polynomial (unsigned int deg);
    Polynomial (unsigned int deg,
                double a[]);
    Polynomial* Add (const
                Polynomial &B);
};

Polynomial::Polynomial
    (unsigned int deg)
{   assert (deg<N);
    n=deg;
    for (int i=0; i<=n; i++)
        Arr[i]=0;
}

Polynomial::Polynomial
    (unsigned int deg,
     double a[])
{   assert (deg<N);
    n=deg;
    for (int i=0; i<=n; i++)
        Arr[i]=a[i];
}

Polynomial* Polynomial::Add
    (const Polynomial &B)
{   int nmax, nmin, i
    if (B.n > n)
        {nmax=B.n; nmin=n;}
    else
        {nmax=n; nmin=B.n;}
    Polynomial* P=new Polynomial
                  (nmax);
    for (i=0; i<=nmin; i++)
        P->Arr[i]=Arr[i]+B.Arr[i];
    if (n==nmax)
        for (i=nmin+1; i<=n; i++)
            P->Arr[i]=Arr[i];
```

```
else
    for (i=nmin+1; i<=B.n; i++)
        p->Arr[i] = B.Arr[i];

return p;
}
```

Another solution for Add:

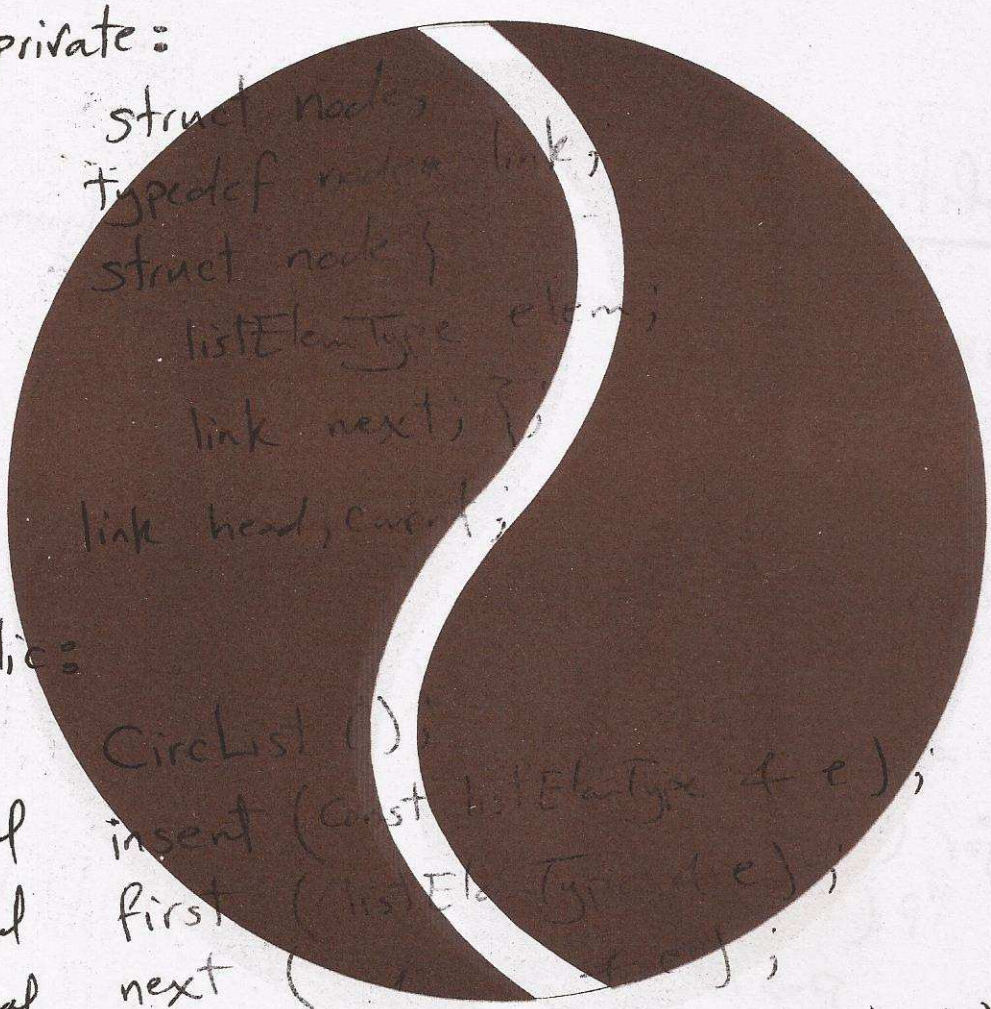
```
Polynomial* Polynomial::Add(Const Polynomial & B)
{
    Polynomial *P; int i;
    if (n > B.n)
    { P = new Polynomial(n);
      for (i=0; i<=n; i++)
        if (i<=B.n)
            P->Arr[i] = Arr[i] + B.Arr[i];
        else P->Arr[i] = Arr[i];
    }
    else { P = new Polynomial (B.n);
           for (i=0; i<=B.n; i++)
             if (i<=n)
                 P->Arr[i] = Arr[i] + B.Arr[i];
             else
                 P->Arr[i] = B.Arr[i];
         }
return P; }
```

Question 2:

→ Definition:

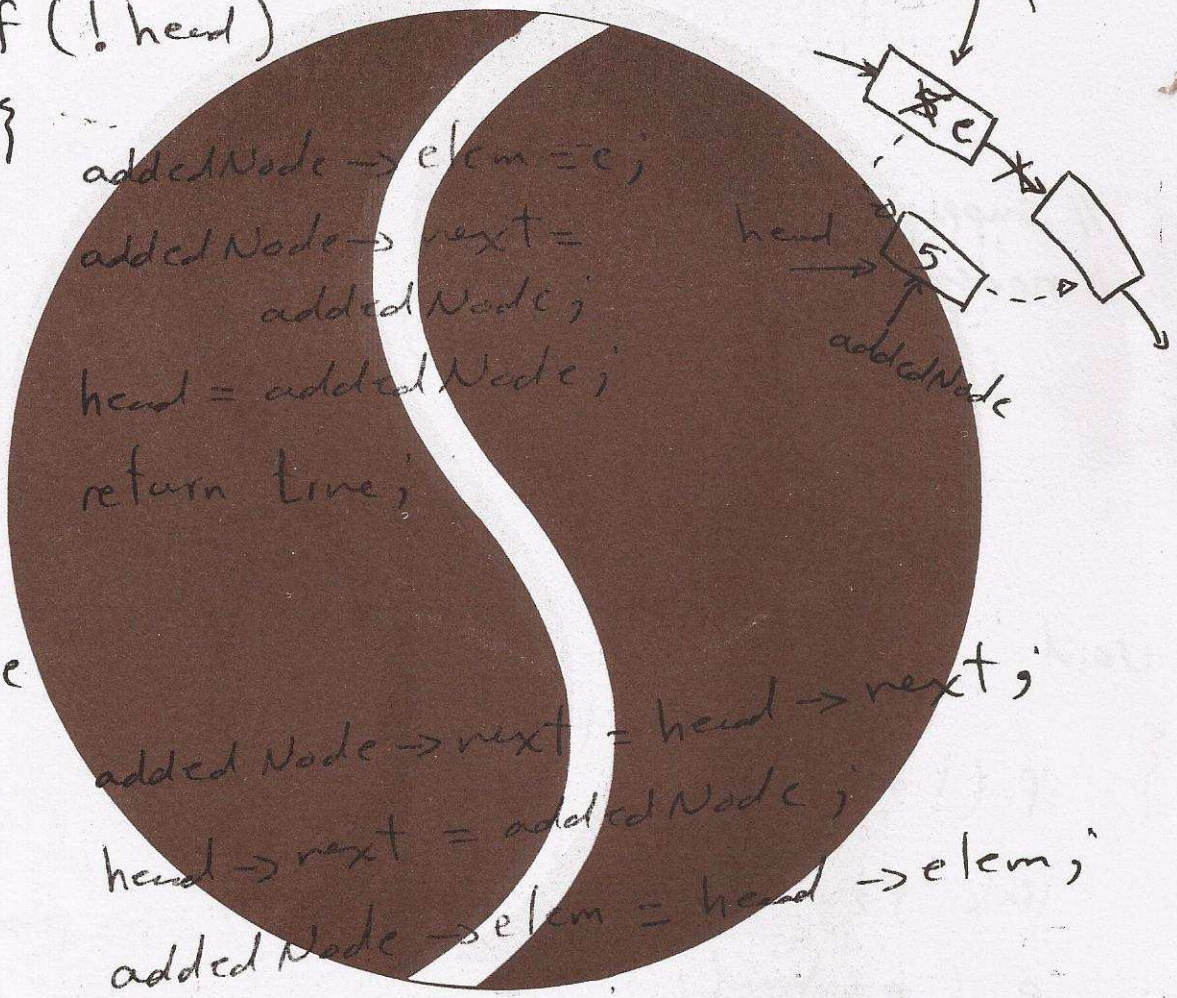
```
typedef int listElemType;
class CircList {
  private:
    struct node;
    typedef node* link;
    struct node {
      listElemType elem;
      link next; };
    link head, curr;

  public:
    CircList ();
    bool insert (const listElemType& e);
    bool first (listElemType& e);
    bool next (listElemType& e);
    bool
    void Merge (const CircList& cList);
    void ReversePrint ();
};
```



→ Functions :

```
bool CircList::insert (const listElemType &e)
{
    link addedNode = new node;
    if (!addedNode) return false;
    if (!head)
    {
        addedNode->elem = e;
        addedNode->next =
                addedNode;
        head = addedNode;
        return true;
    }
else
    {
        addedNode->next = head->next;
        head->next = addedNode;
        addedNode->elem = head->elem;
        head->elem = e;
        head = addedNode;
        return true;
    }
}
```



- Void CircList :: Merge (const CircList & clist)

```
{ if(clist.head==0) return;
  for (link p= clist.head; p->next != clist.head;
                          p= p->next)
      if (! ins
          return;
// supporting two cases of single node or multiple
// nodes
   if (! insert (p->elem)) return;
}
```

- Void CircList :: Rev... Print ()

```
{ if (! head)  retur
  link p; int i, N=1;  i=0;
  for ( p= head; p->next != head; p= p->next)
        N++;
  listElemType * A = new listElemType [N];
  for (p= head; p->next != head; p= p->next)
     {A[i] = p->elem; i++;}
          A[i]= p->elem;
```

```
for (i = N-1; i >= 0; i--)
    cout << A[i];
delete [] A;
}
```

Another Solution:

```
void CircList :: ReversePrint ()
{ if (!head) return;
  Stack <listElemType> S;
  for (link p = head; p->next != head;
                      p = p->next)
      S.push (p->elem);
  S.push (p->elem);
while (!S.isEmpty ())
      cout << S.pop ();
}
```

Third Solution :

```
Void CircList :: ReversePrint ( )
{
   if (!head) return;
   int i, j, N=1;  link p;
   for (p=head; p->next != head; p=p->next)
         N++;

   for (i=N; i>0; i--)
   {
      p=head;
      for (j=1; j<i; j++)
            p= p->next;

      Cout<< p->relem;
   }
}
```

Question 3:

(a)

```
int Replace (Queue &Q, QueueElemType item,
             QueueElemType newValue)
{ int N=0; QueueElemType x;
  Queue Q1;
  while (!Q.isEmpty())
  { x=Q.dequeue();
    if (x==item)
      {Q1.enqueue(newValue); N++;}
    else
      Q1.enqueue(x);
  }
while (!Q1.isEmpty())
    Q.enqueue(Q1.dequeue());

return N;
}
```

b) RPN using stacks.

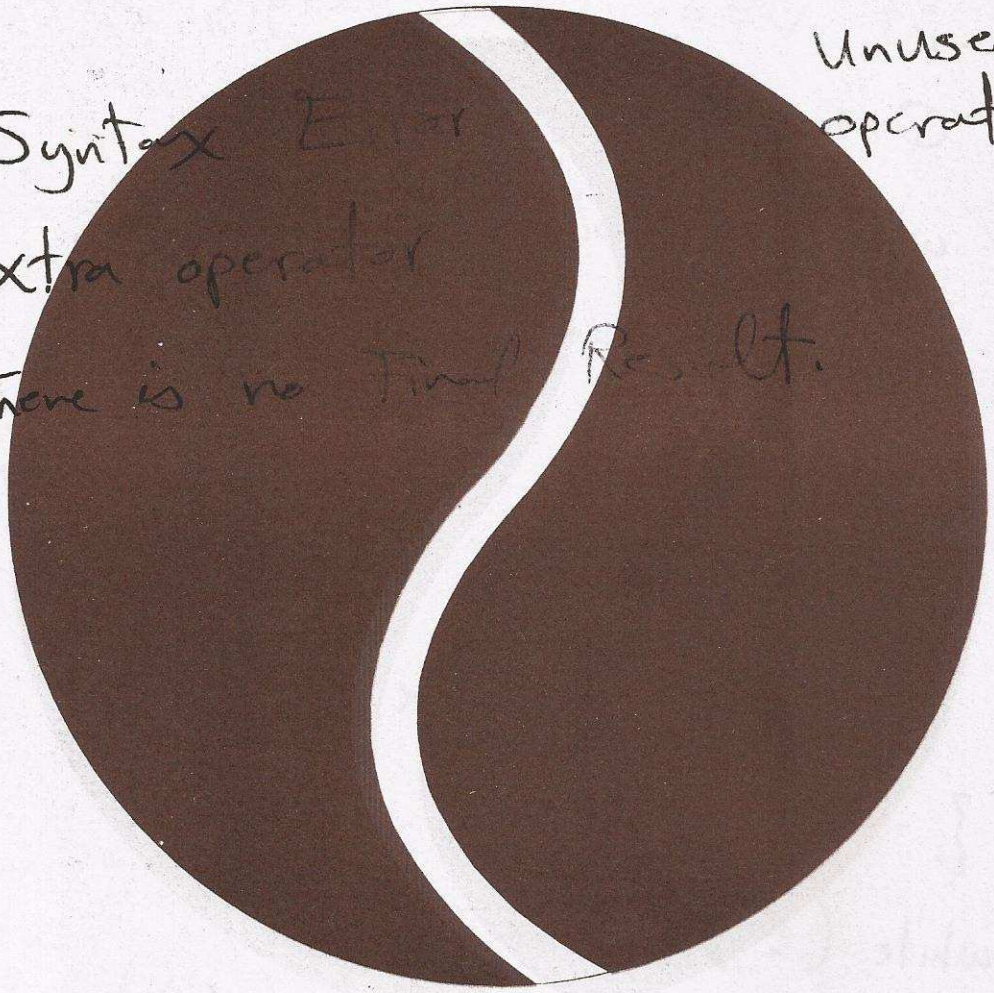
5 3 * 8 - 4 + 3 2 / -

Unused operator

→ Syntax Error

→ extra operator

→ there is no Final Result



Question 5:

a]

i- T, proof of binary search "Canceled"

ii- F, only in worst case "Canceled"

iii- T

iv- T

v- T

b]

i- Linear probing → Table size = 20

key	hash o/p	after probing	Comp.
66	6	6	1
47	7	7	1
87	7	8	2
90	10	10	1
126	6	9	4
140	0	0	1
145	5	5	1
153	13	13	1
177	17	17	1

[30]

285	5	11	7
393	13	14	2
395	15	15	1
467	7	12	6
566	6	16	11
620	0	1	2
735	15	18	4

→ probing o/p = (hash o/p + i) mod 20

where i = 0, 1, 2 ... 19

→ Final Table :

0	140	16	566
1	620	17	197
2	—	18	735
3	—	19	—
4	—		
5	145		
6	66		
7	47		
8	87		
9	126		
10	90		
11	285		
12	467		
13	153		
14	393		
15	395		

no. of Comp.

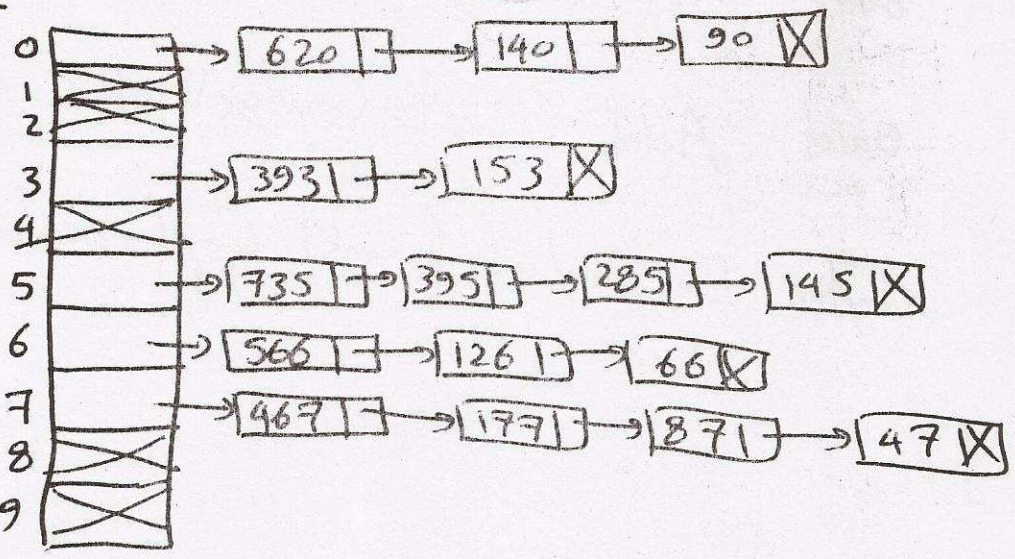
87	2
566	11
153	1

ii- Chained Table:

Table size = 10

Key	hash o/p	no. of Comp
66	6	3
47	7	4
87	7	3
90	0	3
126	6	2
140	0	2
145	5	4
153	3	2
177	7	2
285	5	3
393	3	1
395	5	2
467	7	1
566	6	1
620	0	1
735	5	1

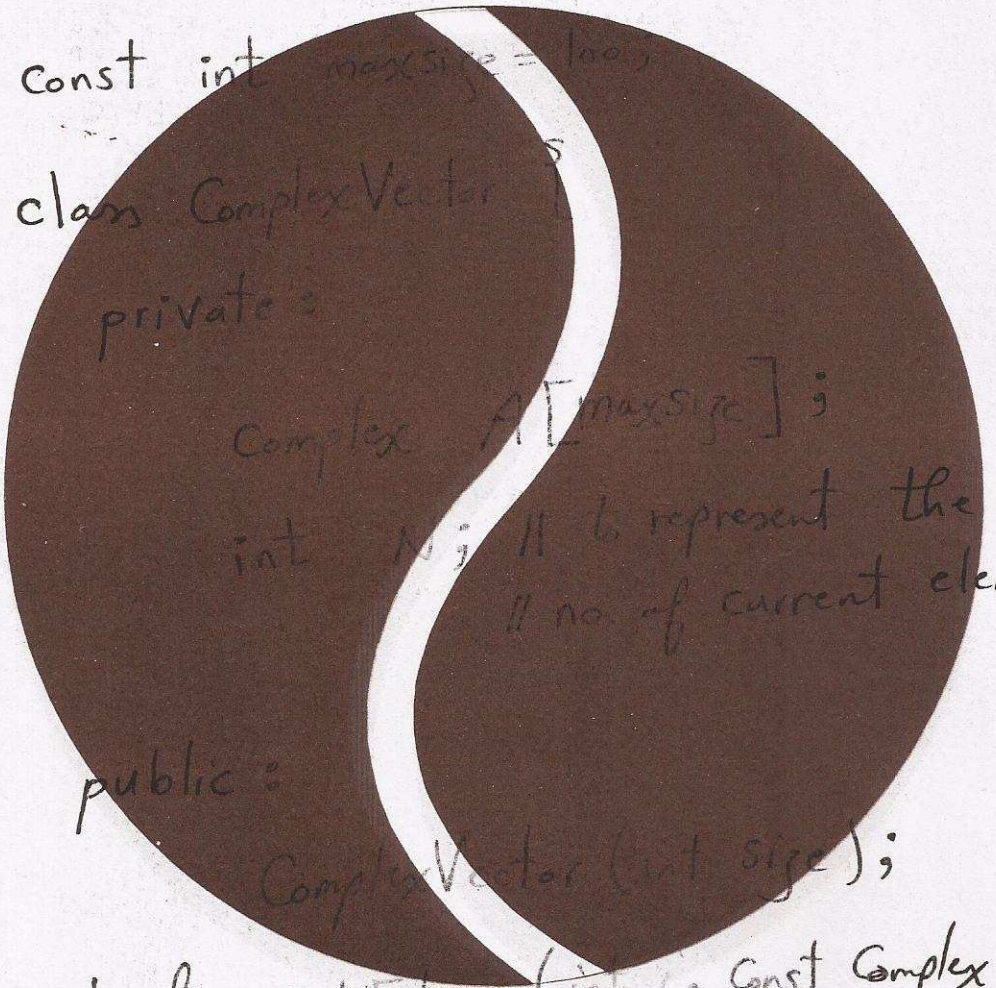
Table:



Final 2011

Q1. b.

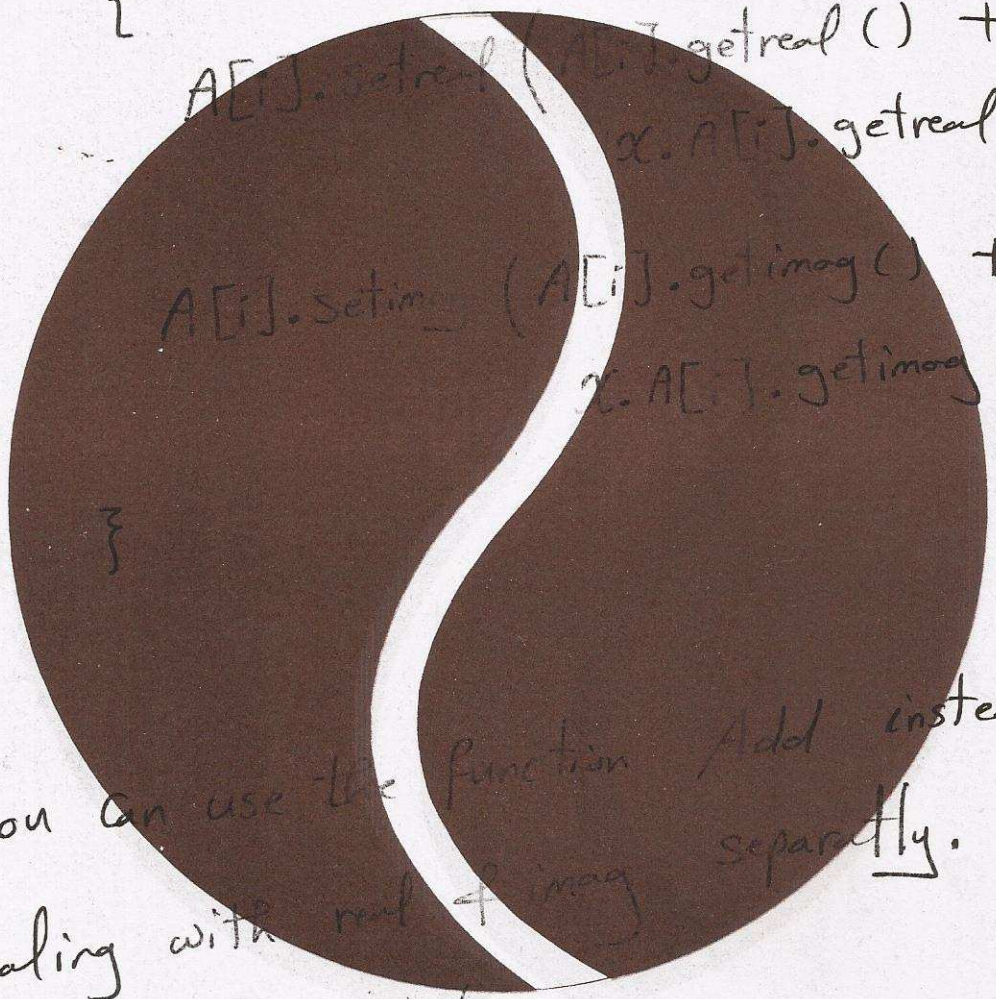
```
#include "Complex.h"
const int maxSize = 100;
class ComplexVector {
private:
    Complex A[maxSize];
    int N; // to represent the
           // no. of current elements
public:
    ComplexVector (int size);
    bool SetElem (int i, const Complex &c);
    void Add (const ComplexVector &x);
    Complex DotProduct (const ComplexVector &x);
};
```



```
ComplexVector :: ComplexVector (int size)
{
    assert (size <= maxsize);
    N = size;
    for (int i=0; i< maxsize; i++)
    {
        A[i].setreal (0);
        A[i].setimag (0);
    }
}

bool ComplexVector :: SetElem (int i, const Complex
                                & c)
{
    if (i >=N)
        return false;
    A[i].setreal ( c.getreal() );
    A[i].setimag (c.getimag() );
    return true;
}
```

```
void ComplexVector::Add(const ComplexVector &x)
{
    assert(N == x.N);
    for(int i=0; i<N; i++)
    {
        A[i].setreal(A[i].getreal() +
                     x.A[i].getreal());
        A[i].setimag(A[i].getimag() +
                     x.A[i].getimag());
    }
}
```

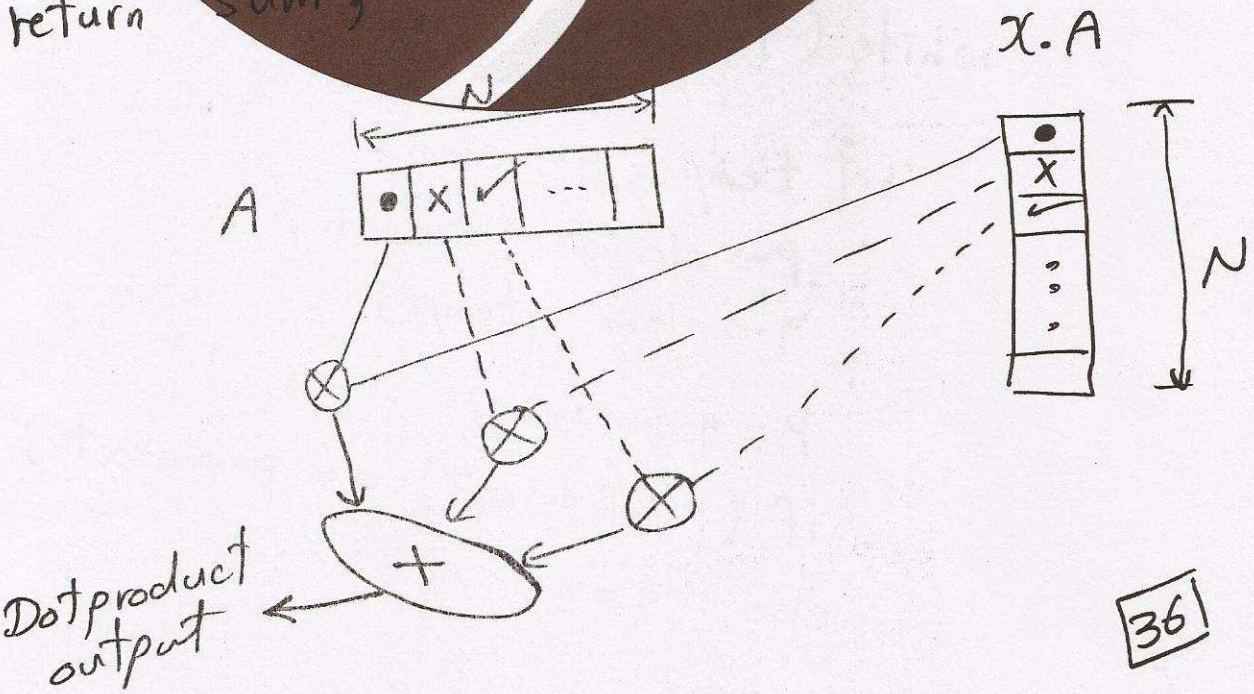


⟹ You can use the function Add instead of dealing with real & imag separately.

So, the loop will be:

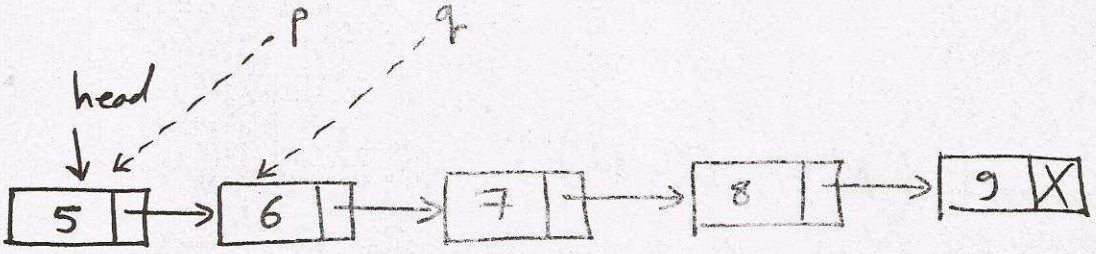
```
for(int i=0; i<N; i++)
    A[i].Add(x.A[i]);
```

```
Complex ComplexVector :: DotProduct (const ComplexVector
                                     &x)
{
    assert (N == x.N);
    Complex sum, temp;
    sum.Setreal (0);   sum.Setimag (0);
    for (int i=0; i<N; i++)
    {
        temp.Setreal (A[i].Getreal ());
        temp.Setimag (A[i].Getimag ());
        temp.Multiply (x.A[i]);
        sum.Add (temp);
    }
    return sum;
}
```

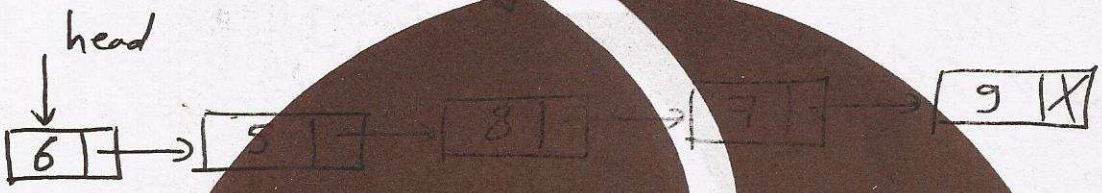


Q2.

9



input

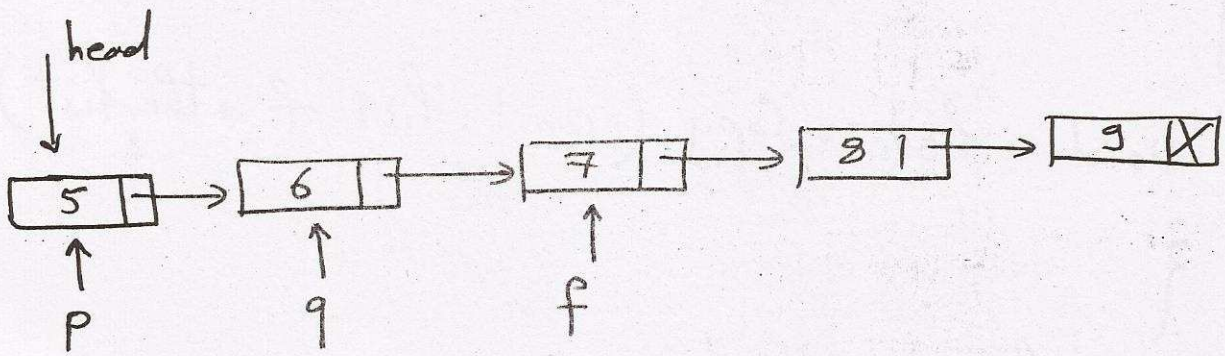


output

```
void list :: interchange ( )
{
if (head == 0 || head -> next == 0) return ;
ListElemType temp ;
link p = head , q = p -> next ;
while (q)
{ temp = p -> elem ;
  p -> elem = q -> elem ;
  q -> elem = temp ;

  p = q -> next ;
  if (p && p -> next) q = p -> next ;
  else q = 0 ; }
}
```

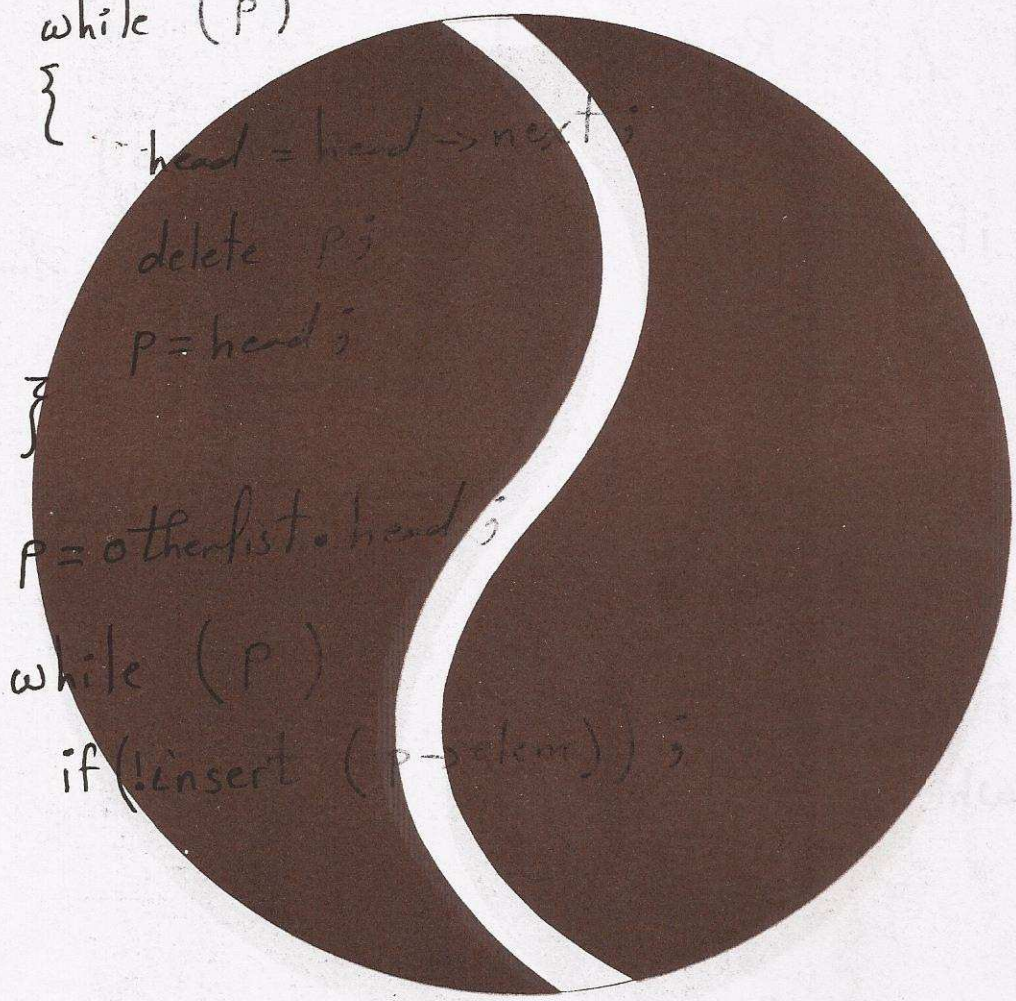
b




```
void list :: Reverse ( )
{
    if (head ==0 || head -> next ==0) return;
    link p = head; link q = p -> next; link f;
    if (q -> next)
        f = q -> next;
    else f = 0;
    p -> next = 0; tail = p;
    while (q)
    {
        q -> next = p;
        p = q;
        q = f;
        if (q)
            f = q -> next;
    }
    head = p;
}
```

c

```
void list::copy (const list & otherlist)
{
    link p = head;
    while (p)
    {
        head = head->next;
        delete p;
        p = head;
    }
    p = otherlist.head;
    while (p)
      if(!insert (p->elem));
}
```



Q3.

a]

```
int push (stackDataType V, int s)
{
    if ( top2 == top1 + 1)
        return -1;
    if (s == 1)
        A[++top1] = V;
    else
        A[--top2] = V;
    return s;
}
```

b]

```
bool DelElement (Queue *Q, int n)
{  Queue T; queueElemType x;
   while (n >= 0 && !Q.isEmpty())
      { x = Q.dequeue();
        if (n != 0) T.enqueue (x);
        n--; }
   while (!T.isEmpty()) Q.enqueue (T.dequeue())
   return (n == -1);
}
```

A General Test

Question 1:

Define an abstract data type, the class `ComplexMatrix`, that represents a square matrix of Complex numbers. Each element of the matrix is an object of the `Complex` ADT. The `Complex` ADT supports the operation `Add`, `Multiply`, and the `Set/Get` functions. The `ComplexMatrix` class contains upto 100 complex numbers per dimension and supports the member functions shown in the table below. Write the definition of the `ComplexMatrix` class and implement its member functions. You need to select the proper return types for all member functions. You don't need to write the `Complex` class ADT.

i) ComplexMatrix (unsigned int dimn)	Constructor to set the dimension of the matrix and initialize all elements to zeros. Dimn should be less than 100.
ii) SetElem (int i, int j, const complex &c)	Inserts the complex number c to the indices (i,j). You should assert for valid indices.
iii) Multiply (const ComplexMatrix &B)	Multiplies the current matrix with matrix B [redacted]. You should assert for the condition of multiplication first.

Question 2:

i. To the standard doubly linked list add the function sort () to sort the list using bubble sort algorithm.

ii. Write a member function split for the circular linked list class. The function splits the current list into two lists such that the first n items are placed in L1 (if possible) and the second n items are placed in L2 (if possible). The current list maintains the remaining items. Note that L1 and L2 are two empty circular lists.

Question 3:

i. The following code is used to sort an array, the algorithm is recursive and it is called quick sort. First and last represent the indexes of the first and last elements in the array.

```
int partitioning (int a[], int first, int last)
{
        int pos=first;
        for (int i=first+1;i<=last;i++)
                if (a[fisrt]>a[i])
                {
                        pos++;
                        swap(a,i,pos);
                }// end of loop
        swap(a,first,pos);
        retrun pos;
}// end of partitioning
```

```
void swap (int a[],int pos1,int pos2)
{
        int temp=a[pos1];
        a[pos1]=a[pos2];a[pos2]=temp;
}
void sort (int a[],int first,int last)
{
        if (first>=last) return;
        int pivot=partitioning(a,first,last);
        sort(a,pivot+1,last);
        sort(a,first,pivot-1);
}
```

a) Trace the above code for the following array 12 5 20 -3 9

b) What is the number of comparisons in the worst case of this algorithm?

ii. Write a class BinarySearch to search for certain target in an array of strings recursively.

Question 4:

i. Write a function deloccurrence (stack & s, stackelemtype v) to delete all occurrence of the element v from the stack s

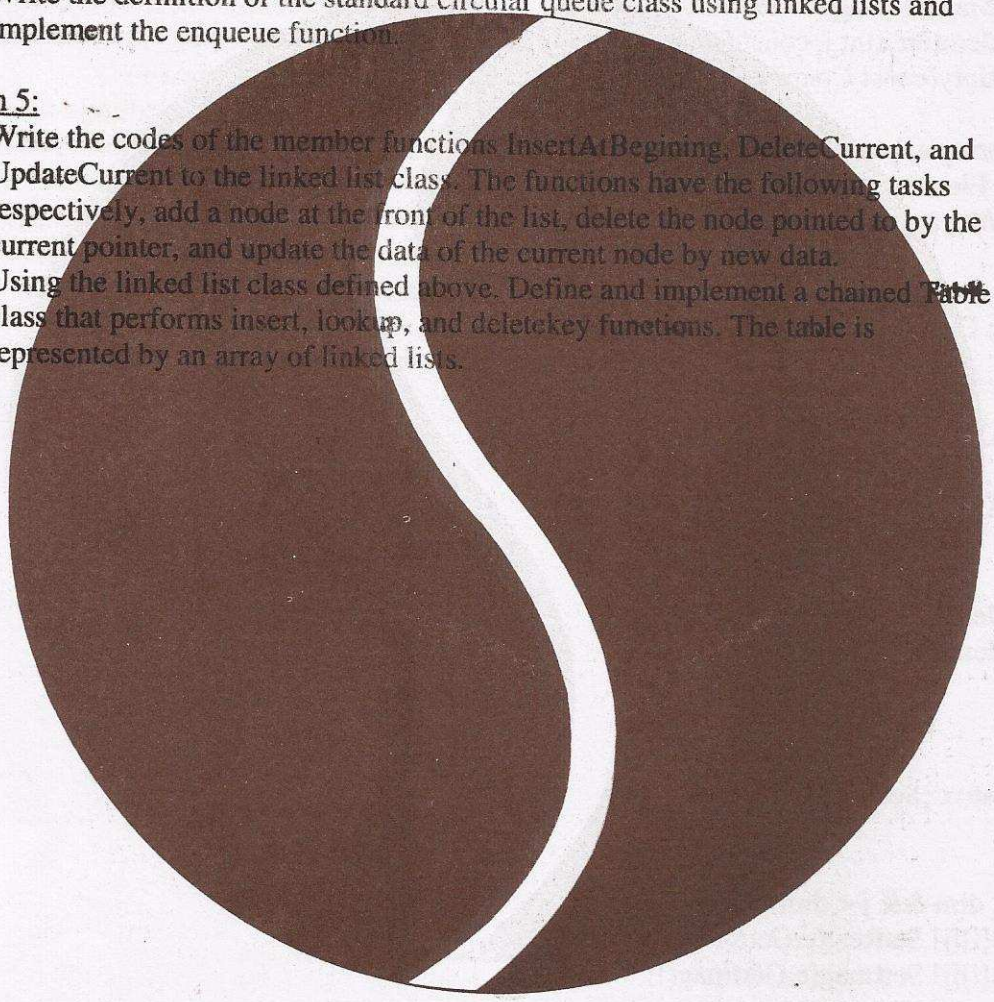
ii. Write a function printprime, to print all prime numbers stored in a given Queue Q.

iii. Write the definition of the standard circular queue class using linked lists and implement the enqueue function.

Question 5:

i. Write the codes of the member functions InsertAtBegining, DeleteCurrent, and UpdateCurrent to the linked list class. The functions have the following tasks respectively, add a node at the front of the list, delete the node pointed to by the current pointer, and update the data of the current node by new data.

ii. Using the linked list class defined above. Define and implement a chained Table class that performs insert, lookup, and deletekey functions. The table is represented by an array of linked lists.



Test solution

Question 1:

```
#include "complex.h"
const int MAXSIZE = 100;
Class ComplexMatrix {
public:
        ComplexMatrix(unsigned int dimn);
        void SetElem(int i,int j, const Complex &c);
        void Multiply(const ComplexMatrix &B);
private:
//the array for storing elements of the vector
        Complex Elements[MAXSIZE][MAXSIZE];
        int dim; //the actual size of the matrix
}

ComplexMatrix:: ComplexMatrix (unsigned int dimn)
{
int i,j;
assert(dimn<MAXSIZE);
dim = dimn ;
for(i=0; i < MAXSIZE; i++)
        for(j=0; j < MAXSIZE; j++)
        {
                Elements[i][j].SetReal(0.0);
                Elements[i][j].SetImag(0.0);
        }
}

void ComplexMatrix::SetElem(int i, int j, const
Complex &c)
{
        assert(i < dim && j < dim);
        Elements[i][j].SetReal(c.GetReal());
        Elements[i][j].SetImag(c.GetImag());
}
```

```
ComplexMatrix ComplexMatrix::Multiply(const
                        ComplexMatrix &B)
{
        assert(dim==B.dim); int i,j,k;
        ComplexMatrix Result(dim);
        Complex temp;
        for (i=0;i<dim;i++)
                for (j=0;j<dim;j++)
                        for (k=0;k<dim;k++)
{
temp= Elements[i][j].Multiply(B.Elements[j][k]);
Result.Elements[i][k]=
Result.Elements[i][k].Add(temp);
}
return Result;
}
```

Question 5:

i- The code of the list will be:

```
#include<assert.h>
template <class ListElementType>
class List
{
public:
List();
void insert(const ListElementType & elem);
bool first(ListElementType & elem);
bool next(ListElementType & elem);
void UpdateCurrent(const ListElementType & e);
void InsertAtBeginning(const ListElementType & e);
void deleteCurrent();
private:
struct Node;
typedef Node *Link;
struct Node
{
ListElementType elem;
Link next;
};
Link head;
Link tail;
Link current;
};
template <class ListElementType>
List<ListElementType>::List()
{
head = 0;
tail = 0;
current = 0;
}
/////first(), next(), and insert() will be the same. We only put [ template <class
ListElementType> ] before
each function and the class name used is [ List<ListElementType> ] like:
template <class ListElementType>
void List<ListElementType>::insert(const ListElementType & elem)
//// member functions that we add:
template <class ListElementType>
void List<ListElementType>::UpdateCurrent(const ListElementType & e)
{
assert(current);
current->elem=e;
```

```
}
template <class ListElementType>
void List<ListElementType>::InsertAtBeginning(const ListElementType & e)
{
Link addedNode=new Node;
addedNode->next=head;
addedNode->elem=e;
head=addedNode;
}
template <class ListElementType>
void List<ListElementType>::deleteCurrent()
{
assert(current);
if(head==0)return; // list is empty
Link del; // if current is at head
if(head==current)
{
del=current;
head=current->next;
current=current->next;
delete del;
return;
}
// current is not at the first node
for(Link p=head;p->next!=current;p=p->next)
del=current;
p->next=current->next;
current=current->next;
delete del;
}
```

ii- The code for the table will be:

```
#include"list.h" //(note that we will use a templated linked list class)
const int MAX_TABLE = 11;
template < class tableKeyType, class tableDataType>
class Table
{
public:
// Table(); we don't need a constructor, since the constructor of list class will cause each
head, tail and
// current to be=0
void insert(const tableKeyType & key, const tableDataType & data);
bool lookup(const tableKeyType & key, tableDataType & data);
void deleteKey(const tableKeyType & key);
void dump();
private:
```

```
struct Slot
{
tableKeyType key;
tableDataType data;
};
List<Slot> T[MAX_TABLE]; // array of linked lists
int hash(const tableKeyType & key);
};
template <class tableKeyType, class tableDataType>
int Table<tableKeyType,tableDataType>::hash(const tableKeyType & key)
{
return key % MAX_TABLE;
}
template < class tableKeyType, class tableDataType >
void Table<tableKeyType,tableDataType>::insert(const tableKeyType & key, const
tableDataType & data)
{
int pos(hash(key));
Slot s,s1;
s1.key=key;s1.data=data;
bool f=T[pos].first(s);
if(f==false)T[pos].insert(s1); //if the list is empty
else
{
while(f)
{ // if the key exists, update slot
if(s.key==key){T[pos].UpdateCurrent(s1);return;}
f=T[pos].next(s);
}
// if key doesn't exist, insert slot at beginning
T[pos]. InsertAtBeginning (s1);
}
}


template < class tableKeyType, class tableDataType >
bool Table<tableKeyType,tableDataType>::lookup(const tableKeyType & key,
tableDataType & data)
{
int pos(hash(key));
Slot s; bool b;
b=T[pos].first(s);
while(b)
{
if(s.key==key)
{data=s.data;return true;}
```

```
b=T[pos].next(s);
}
return false;
}
```

In the previous function, we hash the key, traverse the list comparing the keys. Remember that s is passed by reference to first () and next() so after the call they contain an element (a slot) from the list.

```
template < class tableKeyType, class tableDataType >
void Table < tableKeyType, tableDataType >::deleteKey(const tableKeyType & key)
{
int pos(hash(key));
Slot s; bool b;
b=T[pos].first(s);
while(b)
{
if(s.key==key)
{T[pos].deleteCurrent();return;}
b=T[pos].next(s);
}
}
// deleteCurrent() is a member function that we will add to the linked list class that will
delete the node to
which 'current' points.
template < class tableKeyType, class tableDataType >
void Table<tableKeyType,tableDataType>::dump()
{
Slot s;bool b;
for (int i=0; i < MAX_TABLE; i++)
{
cout << i << '\t';
b=T[i].first(s);
while(b)
{
cout<<s.key<<'\t';
b=T[i].next(s);
}
cout << '\n';
}
cout << '\n';
}
```

Question 2:a

```
void list :: sort ()
{
   if(head ==0 || head-> next ==0)
            return;
   link t =head;
   while (t->next)
        t= t->next;
   link p2=t;  link p1 = t->prev;  listElemtype temp;
   while (t->prev !=0)
   {
      while (p1 !=0)
      {  if( p1->elem > p2->elem)
         { temp = p1->elem;
           p1->elem = p2->elem;
           p2->elem = temp;
         }
      p2=p1;
      p1 = p1->prev;
      }
   t= t->prev;  p2=t;
   if (p2->prev)   p1 = p2->prev;
   }
}
```